Finding Fastest Paths on A Road Network with Speed Patterns

Evangelos Kanoulas, Yang Du, Tian Xia, Donghui Zhang

Northeastern University
Boston, USA
Outline

- Motivation
- Problem Definition
- Related Work
- Query Processing
- Travel-Time Estimator
- Conclusions
Query: From Alice’s home to school, which path is the fastest?

Existing GIS systems
- either find shortest path, not consider the speed at all
- or assume constant speed, such as MapPoint
Motivation

- In real world
  - Highway: 1 hour (10 mph) in rush hours, 10 minutes (60 mph) otherwise
  - Local road: 20 minutes (45 mph) anytime
  - Speed and time do change the fastest path

- Our goal
  - Consider the speed on a road as a function of time
  - Answer queries within a departure interval
Outline

- Motivation
- **Problem Definition**
- Related Work
- Query Processing
- Travel-Time Estimator
- Conclusions
Daily Speed Pattern on A Road

- The real pattern on a road is a continuous function of time.
- The approximation is a piecewise linear constant function of time.
CapeCod Pattern

- Categorized Piecewise Constant speed pattern
  - Categorized (e.g. weekday, weekend)
  - Each day belongs to exactly one category

- The CapeCod pattern of the highway
  - Weekend: [00:00-24:00) 60mph
  - Weekday: [07:00-09:00) 10 mph, otherwise 60mph
CapeCod Network

- Model the road network as a directed graph
- Each edge (a road segment) is associated with a CapeCod pattern
- Adopt the CCAM storage model [Shekhar & Liu, 1997]
  - Use a $B^+$-tree to index the nodes by the Z-ordering (Hilbert values)
Fastest Path Queries

- Given a CapeCod network, a start node $s$ and an end node $e$, and a departure time interval $I$

- Two types of queries:
  - SingleFP
  - AllFP
SingleFP Queries

- Find a single departure time $l_o$ within $I$ and the corresponding fastest path to minimize the travel time

- For example:
  - Alice may leave for work any time in $[7\text{am}, 9\text{am}]$; please suggest the fastest path, e.g. route A, and the corresponding leaving time, e.g. leave at 7:13am
AllFP Queries

- Find the fastest paths for all departure time during $I$
- For example:
  - Alice may leave for work any time in [7am, 9am]; please suggest all fastest paths, e.g. take route A if the leaving time is between 7 and 7:45, and take route B otherwise
- The answer to AllFP query contains the answer to the counterpart SingleFP query
Outline

- Motivation
- Problem Definition
- **Related Work**
- Query Processing
- Travel-Time Estimator
- Conclusions
Related Work

- Shortest Path
  - Classical solutions: Dijkstra’s Algorithm, A* Algorithm
  - Can be straightforwardly extended to solve fastest path problem with constant speed

- Fastest Path with Changing Speed
  - Discrete Time Model
  - Continuous Time Model
The A* Algorithm

- Each node $n_i$ has an estimated distance to the end node $d_{est}(n_i, e)$
- If the estimation is a lower-bound, it is guaranteed to find the shortest path
- A simple estimation is Euclidean distance
The A* Algorithm (cont.)

- Expand the nodes from $s$
- Each time expand the node which has the minimal $d(s, n_i) + d_{est}(n_i, e)$
- The nodes on the other direction, such as $n_2$, may never be expanded
Extend A* to Fastest Path

- Replace distance by travel time (function)

- The travel time along a path can be accurately computed

- Use the lower bound estimation of travel time from $n_i$ to $e$: $\frac{d_{eu}(n_i;e)}{v_{max}}$
Challenges

- Travel time to each node is a function of departure time
  - Which node do we pick to expand?
  - How do we expand the nodes (and the travel time function)?

- The lower bound estimation is too inaccurate
  - How to get better estimation?
Existing Work on Fastest Path

- [Nac95]
  - Discrete time model
  - Apply the A* algorithm for every time instance

- [Cha98]
  - Discrete time model
  - Based on the assumption that the travel time on an edge remains constant after some time

- [SBSP00]
  - Piecewise constant speed model
  - Address queries with a given departure time
Outline

- Motivation
- Problem Definition
- Related Work
- Query Processing
- Travel-Time Estimator
- Conclusions
Overview

- Extend the A* Algorithm
- Maintain a piecewise-linear function for each expanded path in a priority queue
- The function is the sum of the accurate travel time from start node and the estimated travel time to end node
- Pick the path whose minimum value of the maintained function is the smallest
- Maintain a lower border function for all expanded paths
Running Example

- Query time interval [6:50-7:05]

- s → e: [6-8):1/3 mpm
- s → n: [6-7):1/3 mpm, [7-8):1 mpm
- n → e: [6-7:08):1/3 mpm, [7:08-8):1/10 mpm

Graph: 2 miles from s to e, 2 miles from s to n, 1 mile from n to e.
Travel Time on A Road

- Travel time on a road with length $d$
  
  \[
  \begin{align*}
  &\frac{d}{v_1}, \quad \text{if } l \in [t_1, t_2 - \frac{d}{v_1}) \\
  &\left(1 - \frac{v_1}{v_2}\right)(t_2 - l) + \frac{d}{v_2}, \quad \text{if } l \in [t_2 - \frac{d}{v_1}, t_2]
  \end{align*}
  \]
Travel Time from $s$ to Its Neighbors

- $T(l, s \rightarrow e) = 6\text{min}$
  
  $\begin{cases} 
  8 & \text{if } l \in [6:50-6:54) \\
  \geq 6, & \text{if } l \in [6:54-7:00) \\
  \end{cases}$

- $T(l, s \rightarrow n) = \begin{cases} 
  \frac{2}{3}(7:00 - l) + 2, & \text{if } l \in [6:54-7:00) \\
  2, & \text{if } l \in [7:00-7:05] \\
  \end{cases}$
Functions in The Queue

- $T_{est}(n \rightarrow e) = 1 min$
- Expand path $s$ to $n$ first
Before Expanding Node $n$

- Find the departure time interval of node $n$: [6:56, 7:07]
- Compute the travel time from $n$ to $e$ during this interval
Expanding Node $n$

- Compound the travel time function of path $s \rightarrow n \rightarrow e$ and $n \rightarrow e$.
The SingleFP Query Result

- Path $s$ to $n$ to $e$ has the global smallest minimum value
- Report this path and interval $[7:00, 7:03]$ as the answer
The AllFP Query Result

- In this case, return the lower border function.
- In general cases, more paths may be expanded.
Outline

- Motivation
- Problem Definition
- Related Work
- Query Processing
- **Travel-Time Estimator**
- Conclusions
Travel-time Estimator

- The more accurate the estimation, the more efficient the algorithm.

- Euclidean distance divided by maximum speed is very inaccurate in most cases.

- We propose *boundary-node estimator*.
  - Cell partitioning
  - Pre-computation
Boundary-node Estimator

- Space is divided into non-overlapping cells.

- Boundary node: a node directly connected to a node in other cell.

- Pre-computation:
  - For each pair of cells, precompute the shortest path between their boundary nodes.
  - For each node, precompute the shortest path from and to the boundary nodes.
Boundary-node Estimator (cont.)

- $b_1$ to $b_2$ is the shortest path between two cells
- $b_3$ is the nearest boundary node from $n$, and $b_4$ is the nearest to $e$
- Estimation: $d_{est}(n, e) = d(n, b_3) + d(b_1, b_2) + d(b_4, e)$
Outline

- Motivation
- Problem Definition
- Related Work
- Query Processing
- Travel-Time Estimator
- Conclusions
Conclusions

- Proposed the CapeCod pattern, which captures the real-life speed information, and two practical queries.

- Proposed an algorithm to answer both queries based on novel extensions to A* algorithm.

- Provided a new lower-bound estimator to improve the efficiency.
Thank you!

Q & A